

Data Streams

Internet Analytics (COM-308)

Prof. Matthias Grossglauser
School of Computer and Communication
Sciences



Overview

- Performance criteria:
 - Data volume
 - Computational cost
- We have paid some attention to this:
 - Examples: SGD vs “full” gradient descent; power method for PageRank
- But: assumption that we always have access to all the data!
 - Allows iterative algorithms
 - Many algorithms require “random access”
- Data streams:
 - Relax these assumptions in different ways
 - Tradeoff: compute simple quantities very efficiently

Motivating example (1)

- Internet backbone router
- Order of magnitude:
 - 100s of interfaces at 10s of Gbps
 - = several billion pkts/sec!
 - ...at expensive SRAM speeds!
- Traffic analysis app to detect DDoS attack:
 - What are the **dominant** flows?
 - How many **different** (unique) source IP addresses in a minute?



Motivating example (2)

- Implantable medical devices:
 - Resource-limited: memory, computation, energy
 - Rare/unpredictable read-outs
- Sensor reads:
 - Storing full trace may not be feasible
 - Extract key statistics & maintain over time



Data stream model

- Computing statistics with sub-linear memory
- Example:
 - n numbers: how many unique values (colors) k ?



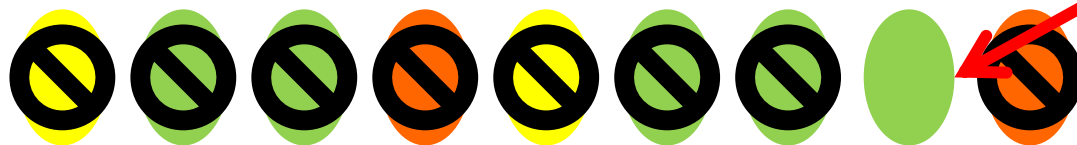
- How to solve with $\theta(n \log n)$ memory?
 - Keep every value in some efficient data structure; compare & count
- How to solve with $o(n)$ memory?
 - Cannot solve exactly
- Streaming algorithms: ∞ data, finite memory
 - Approximation
 - (Pseudo-)randomization

Estimates with sublinear memory

- Counting # of elements:
 - $O(\log n)$ space
- Maximum value:
 - $O(1)$ space
- Average value:
 - Sum / counter: $O(\log n)$ space
- Heavy hitters:
 - Find the top values that repeat the most
 - Table of all (values, counts) seen so far has size $O(n \log n)$
- Number of distinct elements?
 - I.e., do not double-count multiple occurrences
 - Table of values so far: $O(n \log n)$

(1): Heavy hitters

- Find most frequent values in a stream
- Trivial solution: keep counters for every value, sort at the end
 - Cost: space $\Omega(m)$ (m : alphabet size)
- Threshold criterion θ :
 - Report every value a that has occurred $> \theta n$ times
- Case $\theta = 1/2$: majority (at most one value)
- Trick: pairwise annihilation of different observed values

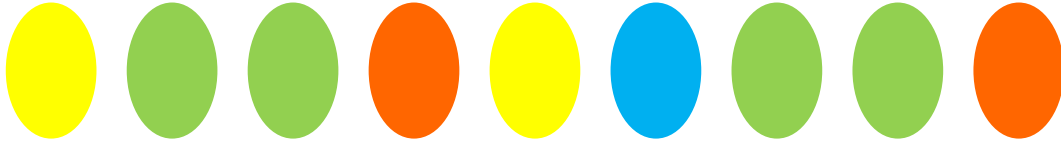


May or may not be majority color \rightarrow 2nd pass to re-tally

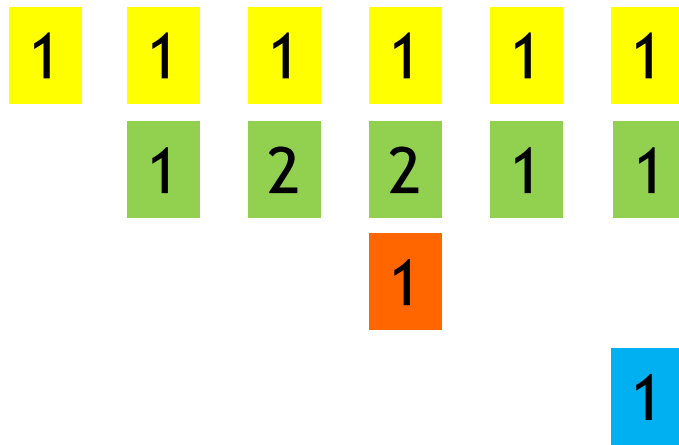
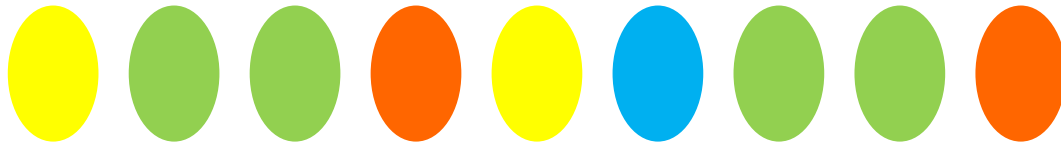
Heavy hitters: algorithm

- For $\theta < 1/2$, need larger “annihilation set” K
- Algorithm:
 - Annihilation set K contains (value, count) tuples
 - For every new sample x , update (or create) tuple in K
 - When $|K| \geq 1/\theta \rightarrow$ decrease all counts by 1, and delete those hitting 0
- Property:
 - At the end, K is a superset of values with frequency θn
 - Second pass needed to get actual counts
- Cost: $O(\log n / \theta)$ space, $O(n)$ complexity

Heavy hitters: example ($\theta = 1/3$)



Heavy hitters: example ($\theta = 1/3$)



correct: (4/9)

wrong: (2/9)

Heavy hitters: proof

- Lemma:
 - A value x that occurs at least θn times in the stream is in K at time n (with count ≥ 1)
- Proof:
 - Def: **decrement action** = decrement all nonzero counters by one
 - Each decrement action decreases the total count (sum of all counters) by at least $1/\theta$
 - Therefore, at most θn decrement actions over dataset
 - Each individual value x gets decremented by at most one for each decrement action \rightarrow at most θn decrements per x
 - If x occurs more than θn times, its counter cannot be 0 at time n

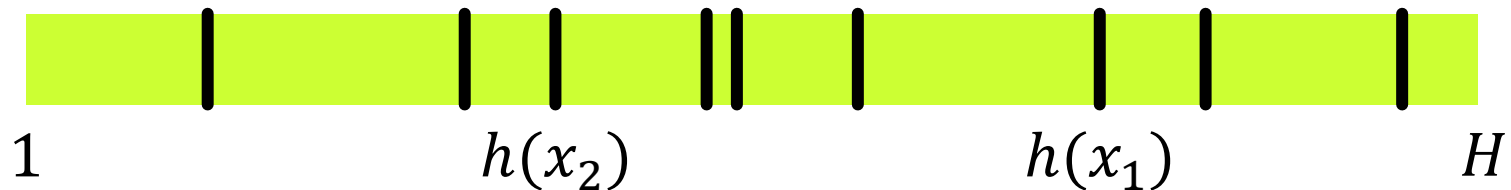
(2) Counting distinct elements



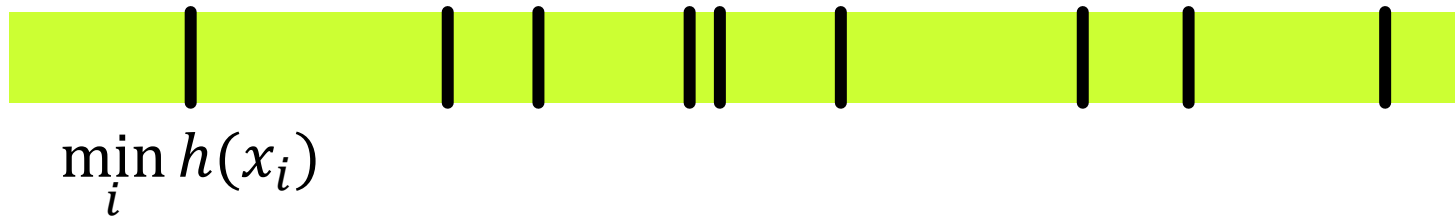
- Number of distinct elements?
 - Table of values seen: $O(n \log n)$
 - Hash table
 - Can we do better?
 - No - if we need exact answer
- Approximation:
 - Many streaming algorithms trade off small loss in precision (approximation) with large gain in memory requirement

Approximate count-distinct

- Flajolet-Martin algorithm
- Intuition:
 - Hash function $h(x)$:
 - Pseudorandom:
 - Random: $x \neq y \rightarrow h(x)$ and $h(y)$ can be regarded as independent uniform random variables
 - Pseudo: $x = y \rightarrow h(x) = h(y)$
 - k distinct values $x_1, \dots, x_k \rightarrow$ hashed values $h(x_1), \dots, h(x_k)$



Count-distinct: single hash function



- Normalize $H = 1$
- What is $y = E[\min_i h(x_i)]$?
 - $k + 1$ intervals $\rightarrow y = (k + 1)^{-1}$
- Now suppose we have n values x_1, \dots, x_n , but not necessarily distinct
- What is $y = E[\min_i h(x_i)]$ now?
 - Still $y = (k + 1)^{-1}$, where k is # of distinct elements
- Therefore $\hat{k} = \frac{1}{\min_i h(x_i)} - 1$ is an estimator for # of distinct elements in x_1, \dots, x_n

Count-distinct: multiple hash functions

- Improving the estimate:
 - Obtain multiple independent estimates & average, by using multiple hash functions:
 - Assumption: $h_1(x)$ and $h_2(x)$ can be regarded as independent RVs



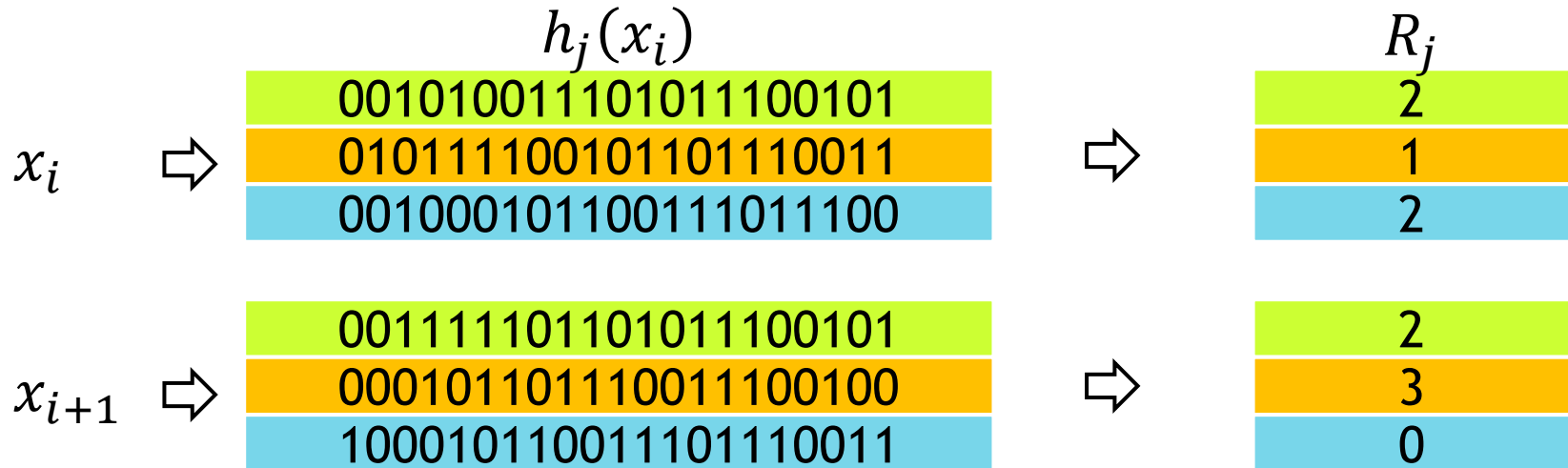
Flajolet-Martin algorithm

- Discretization of minimum estimator \rightarrow compression
- Binary representation:
- $z(x)$ = number of leading 0s (or trailing)
 - Example: 00**1**0101111000101011010 $\rightarrow z = 2$, ie, $x \in \{001000000 \dots, 001111111 \dots\}$



- Maximum # of zeroes:
$$R = z(\min_i h(x_i)) = \max_i z(h(x_i))$$
- Estimate: $\hat{k} = 2^R / 0.77351$
 - Factor ensures that estimate is unbiased despite discretization

FM algorithm: implementation



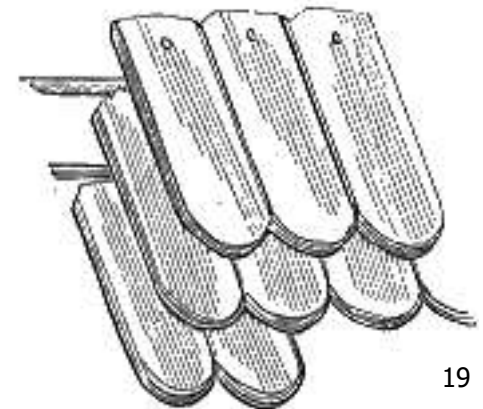
- Combining $\{R_j\}$ into a single estimate for \hat{k} :
 - 2^R has power-law distribution \rightarrow median $\{2^{R_1}, \dots, 2^{R_J}\}$ is better than average
 - (or average of medians over subgroups)

(3) Document similarity

- Given: Corpus of documents
- Efficient way to find pairwise similarity
- Applications:
 - Web crawling: eliminate copies or similar version of documents
 - Filter search results: only show sufficiently different docs
- Solutions:
 - Cosine similarity or similar: too costly for some applications

Document similarity: sketch

- How to compare two docs very efficiently?
- Approach 1: vector space model
 - Two docs are similar if vectors of word frequencies are similar
 - Drawback: bag-of-words approach loses all structure
- Approach 2: use more local structure → shingles (or n-grams)
 - k -shingle: ordered sequence of k consecutive words
 - “bag of shingles” - large, sparse vocabulary
 - Example: “And now for something completely different” → 3-shingles: “and now for”, “now for something”, “for something completely”,...

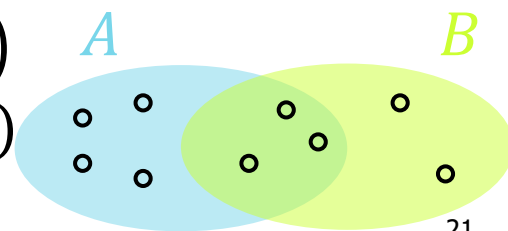


Document similarity: shingles

- Two documents characterized by their sets of shingles A and B
- Similarity: Jaccard index
 - $sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$
- Problem:
 - Vector-space representation: $|V|^k$ -sized vocabulary
 - Sparse representation: most shingles are unique (for k not too small) $\rightarrow \theta(n)$ space, usually much better than $|V|^k$, but still too much for some applications
- Solution:
 - Sketch: compressed version of vector of shingles

Document sketch

- Goal: compact representation of A, B to estimate $\text{sim}(A, B)$
- Hash function $h(.) \in [1, H]$: maps shingle to integer
 - Assume H is large enough (e.g., 2^{64}) to avoid collisions
- Def: sketch $s(A) = \min_{a \in A} h(a)$
- Lemma: $P(s(A) = s(B)) = \frac{|A \cap B|}{|A \cup B|}$
- Proof:
 - Hash function \rightarrow Each element x in $A \cup B$ has same prob. of being min
 - No collision assumption $\rightarrow P(s(A) = s(B))$
 $= P(\text{random } x \in A \cup B \text{ is in } A \cap B)$
 $= |A \cap B| / |A \cup B|$



Document sketch



- To reduce variance: multiple hash functions (e.g., 100-1000)
- $s(A) = (\min h_1(A), \min h_2(A), \dots, \min h_J(A))$
- $\widehat{sim}(A, B) = \frac{|\{s: h_j(A) = h_j(B)\}|}{J}$
 - Unbiased
 - Doc sketch comparison:
 - A few hundred integer comparison operations \rightarrow very efficient

$s(A)$	17	204	94	144	78	12	204	42	...
$s(B)$	87	204	185	91	78	12	84	42	...

(4) Distances and nearest-neighbor

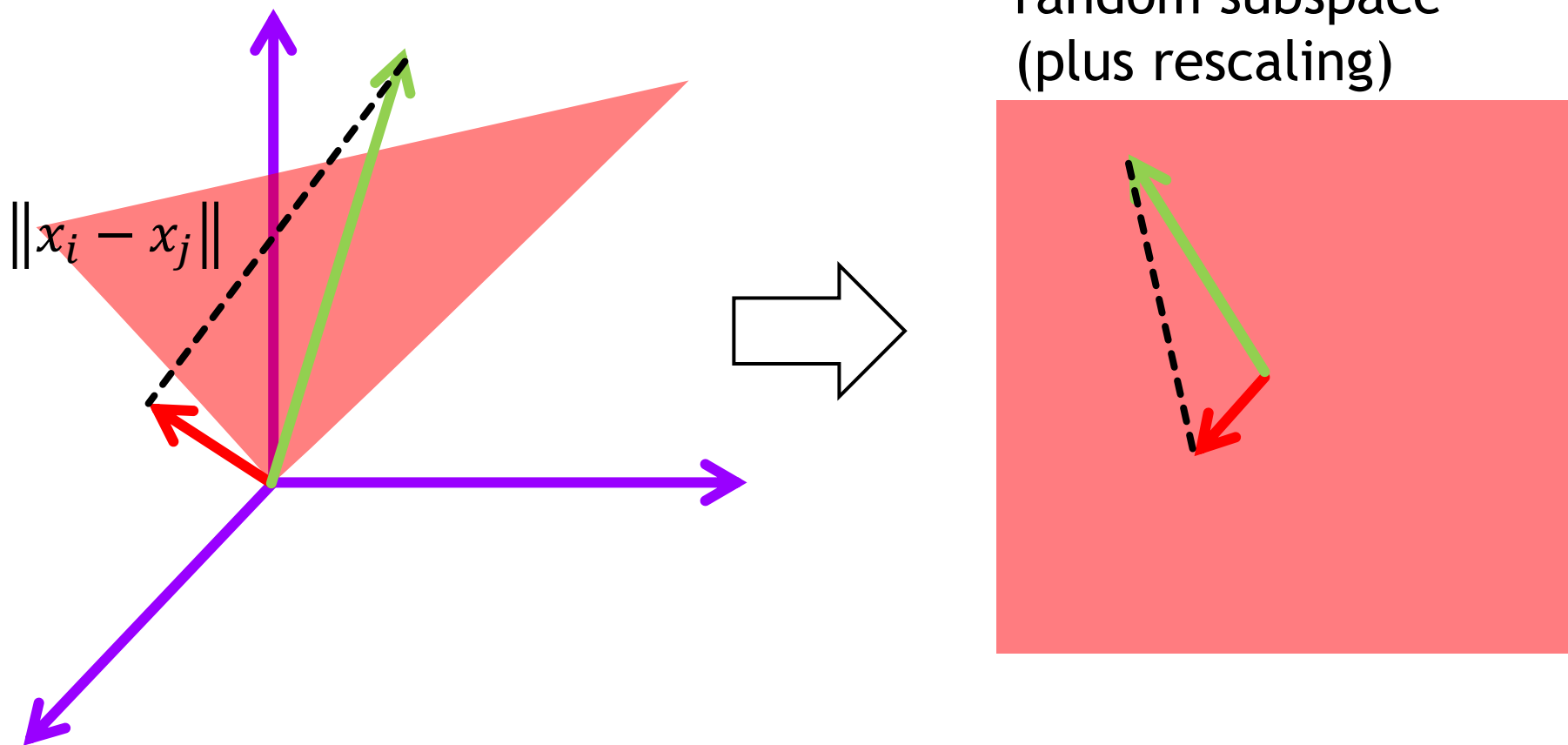
- So far: counts, set similarity
- Another general problem: high-dimensional data $x_i \in \mathbb{R}^d$, with d large
 - Examples: images; time series (financial, sensors,...)
- Euclidean distance $\|x_i - x_j\|$
- Often, one needs nearest-neighbor queries:
 - For a given x , find x_i minimizing $\|x_i - x\|$
 - Examples: find most similar images; find user with similar pref vector in recommender system
- Large d and n : very costly computation
- How can we bring down dimensionality?

Randomized dim reduction

- Johnson-Lindenstrauss Lemma:
 - Given: n points $x_i \in \mathbb{R}^d$, error tolerance $0 < \epsilon < 1$
 - There is a (linear) function $f: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$, such that
 - $(1 - \epsilon) \|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1 + \epsilon) \|x_i - x_j\|^2$
 - for $d' > 8 \ln(n)/\epsilon^2$  Does not depend on d
 -  Very benign growth in n
- Numerical example:
 - $n = 1000$ images of $d = 1\text{m}$ pixels each; suppose $\epsilon = 0.2$
 - $d' \cong 1400$

Random projection

- Techniques: project into a **random** low-dim subspace



Random projection

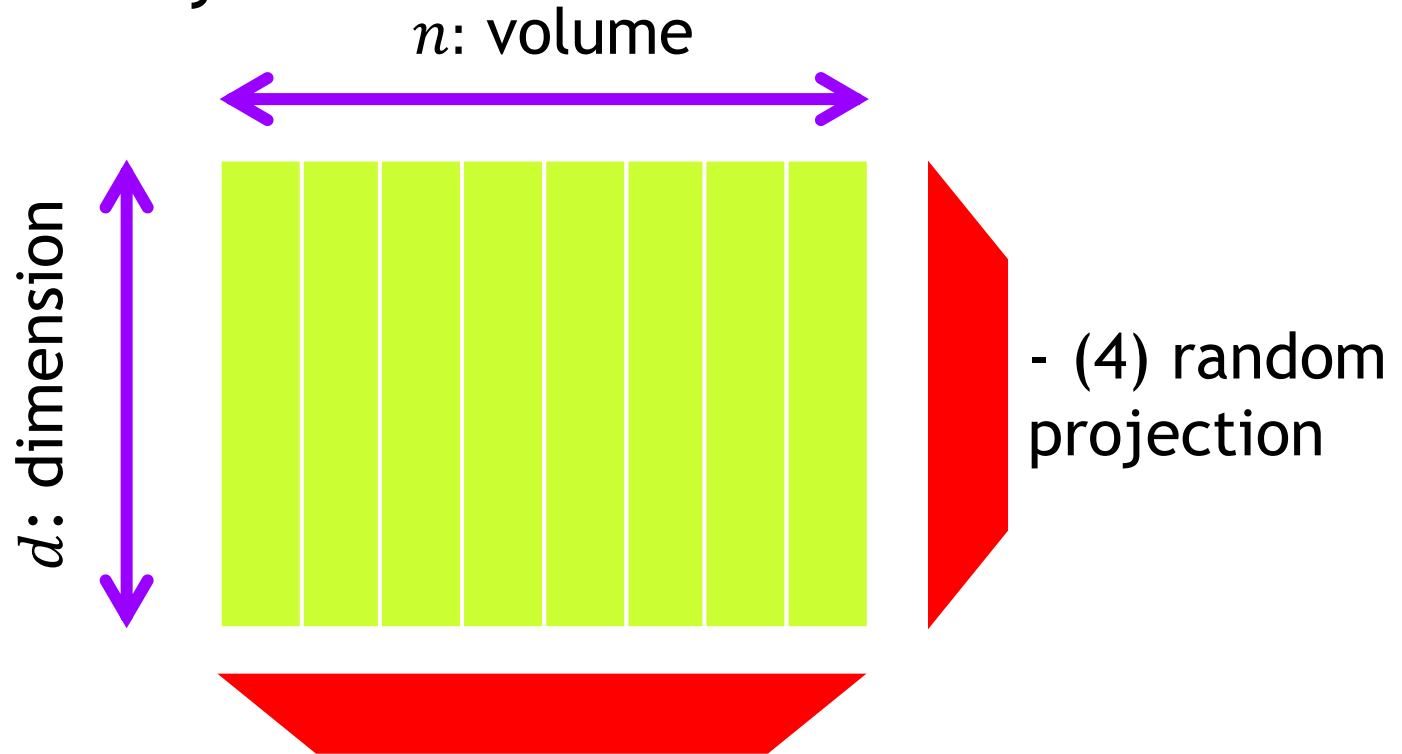
- Intuition: why can we project into a space of dim d' independently of original dim d ?
 - High dimension: almost everything is far away
 - When projecting into d' -dim random subspace:
 - For $\|x_i - x_j\|$ small \rightarrow it stays small
 - For $\|x_i - x_j\|$ large \rightarrow for distance to collapse, d' different dimensions would have to “miss” the difference
 - Probability of this happening drops very quickly with d'
 - $\ln(n)$ factor: price to pay for this to hold for n^2 different pairs

Dim reduction with random projections

- Very efficient way of maintaining distances between large collection of objects
- Version:
 - Random vectors (components restricted to $\{-1, +1\}$)
- Example application:
 - Autonomous security camera taking pictures
 - Cannot transmit all pictures, but want to answer queries of the type “when did things look different?”, or cluster similar scenes

Summary

- “Big Data” challenges: volume and curse of dimensionality



- (1) Heavy-hitter
- (2) Flajolet-Martin sketch
- (3) Doc shingle sketch

References

- [A. Rajaraman, J. D. Ullman: Mining of Massive Datasets, 2012 (chapter 4)]
- [S. Muthukrishnan: Data Streams - Algorithms and Applications, Foundations and Trends in Theoretical CS, 1:2, 2005]